Random Variate Generation by Numerical Inversion When Only the Density Is Known

Derflinger, Gerhard; Hörmann, Wolfgang; Leydold, Josef

Published: 01/09/2009

Document Version
Publisher's PDF, also known as Version of record

Random Variate Generation by Numerical Inversion When Only The Density Is Known

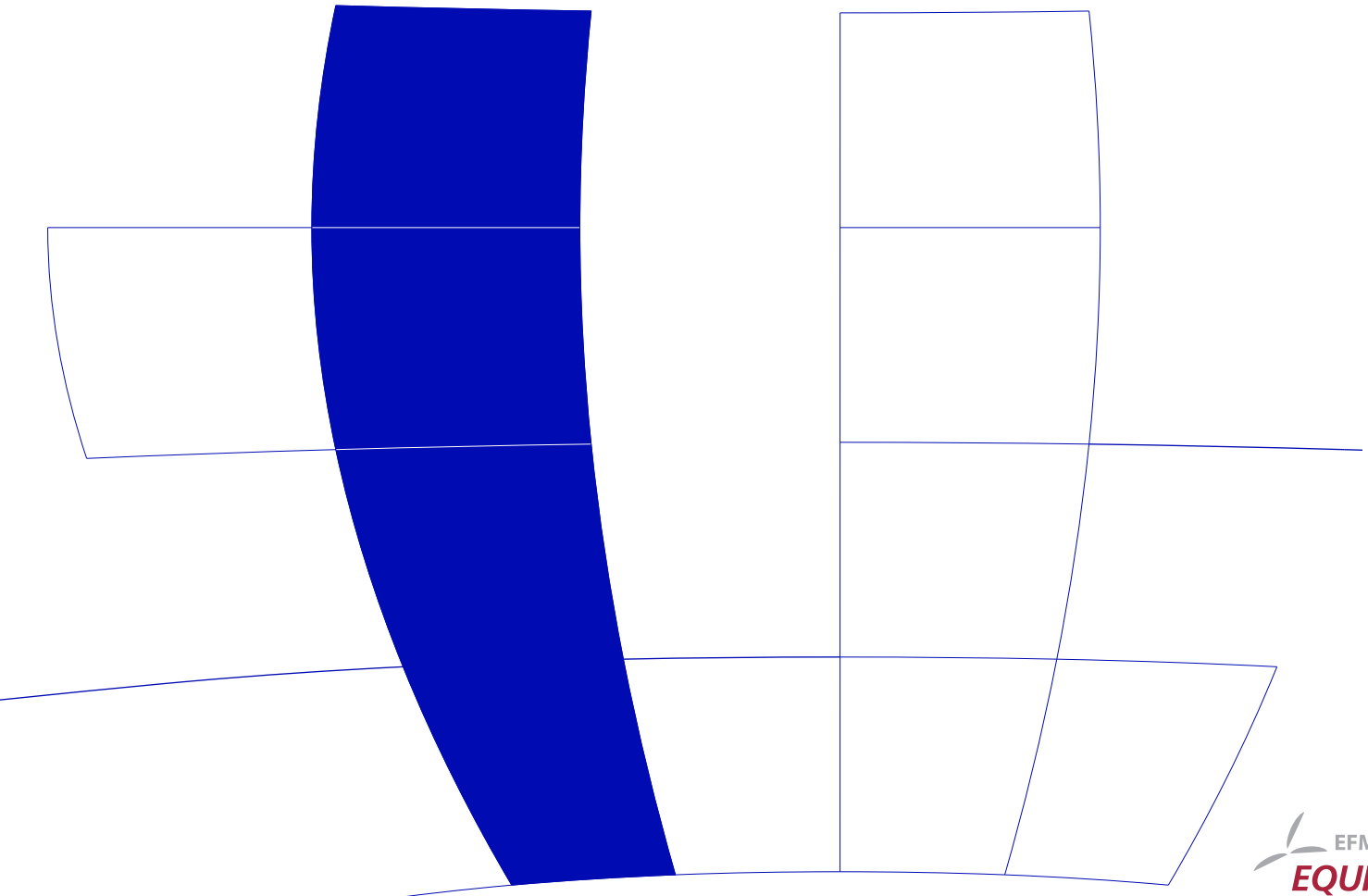
Gerhard Derflinger, Wolfgang Hörmann, Josef Leydold

Department of Statistics and Mathematics
WU Wirtschaftsuniversität Wien

Research Report Series

Report 90
September 2009

http://statmath.wu.ac.at/



Random Variate Generation by Numerical Inversion When Only The Density Is Known

GERHARD DERFLINGER
WU (Vienna University of Economics and Business)
WOLFGANG HÖRMANN
BU Istanbul
and
JOSEF LEYDOLD
WU (Vienna University of Economics and Business)

We present a numerical inversion method for generating random variates from continuous distributions when only the density function is given. The algorithm is based on polynomial interpolation of the inverse CDF and Gauss-Lobatto integration. The user can select the required precision which may be close to machine precision for smooth, bounded densities; the necessary tables have moderate size. Our computational experiments with the classical standard distributions (normal, beta, gamma, t-distributions) and with the noncentral chi-square, hyperbolic, generalized hyperbolic and stable distributions showed that our algorithm always reaches the required precision. The setup time is moderate and the marginal execution time is very fast and nearly the same for all distributions. Thus for the case that large samples with fixed parameters are required the proposed algorithm is the fastest inversion method known. Speed-up factors up to 1000 are obtained when compared to inversion algorithms developed for the specific distributions. This makes our algorithm especially attractive for the simulation of copulas and for quasi-Monte Carlo applications.

Categories and Subject Descriptors: G.3 [**Probability and Statistics**]: Random number generation

General Terms: Algorithms

Additional Key Words and Phrases: non-uniform random variates, inversion method, universal method, black-box algorithm, Newton interpolation, Gauss-Lobatto integration

1. INTRODUCTION

The *inversion method* is the simplest and most flexible method for drawing samples of non-uniform random variates. For a target distribution with given cumulative distribution function (CDF) F a random variate X is generated by transforming

Author's address: Gerhard Derflinger and Josef Leydold: Department of Statistics and Mathematics, WU (Vienna University of Economics and Business), Augasse 2-6, 1090 Vienna, Austria, email: Gerhard.Derflinger@wu.ac.at, Josef.Leydold@wu.ac.at
Wolfgang Hörmann: Department of Industrial Engineering, Boğaziçi University, 34342 Bebek-Istanbul, Turkey, email: hormannw@boun.edu.tr

uniform random variates U using

$$X = F^{-1}(U) = \inf\{x: F(x) \geq U\} .$$

For *continuous distributions* with *strictly monotone* CDF, $F^{-1}(u)$ simply is the inverse distribution function (quantile function). The *inversion method* is attractive for stochastic simulation due to several important advantages:

—It is the most *general* method for generating non-uniform random variates. It works for all distributions provided that the CDF is given (and there is a way to compute its inverse).

—It transforms uniform random numbers U *one-to-one* into non-uniform random variates X.

—It *preserves* the structural properties of the underlying uniform pseudo-random number generator (PRNG).

—It allows easy and efficient sampling from *truncated* distributions.

—It can be used for variance reduction techniques (common or antithetic variates, stratified sampling, ...).

—It is well suited for *quasi-Monte Carlo* methods (QMC).

—It is essential for *copula* methods as one has to transform the uniformly distributed marginals of the copula into the marginals of the target distribution.

Hence it has long been the method of choice in the simulation community (see, e.g., Bratley et al. [1983]) and it is generally considered as the only possible alternative for QMC and copula methods.

Unfortunately, the inverse CDF is usually not given in closed form and thus one must use numerical methods. Inversion methods based on well-known root finding algorithms such as Newton method, regula falsi or bisection are *slow* and can only be speeded up by the usage of often large tables. Moreover, these methods are *not exact*, i.e., they produce random numbers which are only approximately distributed as the target distribution. Despite the importance of the inversion method most simulation software packages do not provide any automatic inversion method; others only provide robust but expensive root finding algorithms (e.g., Brent-Dekker method and the bisection method in SSJ [L'Ecuyer 2008]). An alternative approach uses interpolation of tabulated values of the CDF [Hörmann and Leydold 2003; Ahrens and Kohrt 1981]. The tables have to be precomputed in a setup but guarantee fast marginal generation times which are almost independent of the target distribution. Thus such algorithms are well-suited for the *fixed parameter* case where large samples have to be drawn from the same distribution.

However, often we have distributions where (currently) no efficient and accurate implementation of the CDF is available at all, e.g., generalized hyperbolic distributions and the noncentral χ^2-distribution. Then numerical inversion also requires numerical integration of the probability density function (PDF). The first paper describing a numerical inversion algorithm seems to be Ahrens and Kohrt [1981]. It is based on a fixed decomposition of the interval $(0, 1)$. For each of the subintervals the inverse CDF is approximated by a truncated expansion into Chebyshev polynomials. Ulrich and Watson [1987] compared that algorithm with some other methods where they combined different ready-to-use routines: double precision integration

routine embedded in a Newton root finding algorithm, a packaged ordinary differential equation solver (ODE solver), Runge-Kutta approximation, and polynomial approximation using B-splines. Both references have a major drawback: They do not allow the user to select the size of the maximal acceptable interpolation error.

Due to the advantages listed above fast numerical inversion algorithms are of greatest practical importance. We are convinced that there is need for a paper that describes all numerical tools for such an algorithm and explains the non-trivial technical details necessary to reach high precision. The aim of our research in the last five years was therefore to design a robust black-box algorithm to generate continuous random variates by numerical inversion. The user only has to provide the PDF and a "typical point" in the domain of the distribution (e.g., a point near the mode) together with the size of the maximal acceptable error. We arrived at an algorithm that is based on polynomial interpolation of the inverse CDF utilizing Newton's formula together with Gauss-Lobatto integration. The approximation error can be either calculated during a (possibly very) slow setup, or estimated with a simple heuristic to obtain a faster setup. (That heuristic worked well for all our experiments but the maximal acceptable error is not guaranteed for all PDFs.) Our algorithm is new, compared to the algorithms of [Ahrens and Kohrt 1981; Ulrich and Watson 1987], as it introduces automatically selected subintervals of variable length as well as a control of the error. Compared to Hörmann and Leydold [2003] the new algorithm has the main practical advantage that it does not require the CDF together with the PDF but only the PDF. It also requires a smaller number of intervals, and numerical tests show that the error control is much improved.

The paper is organized as follows. In Section 2 we discuss some principles of numerical inversion, in particular the concept of *u-error* that is used to assess the quality of a numerical inversion procedure. Section 3 describes all ingredients of the proposed algorithm, that is, Newton's interpolation formula, Gauss-Lobatto quadrature, estimation of appropriate cut-off points for tails, and a method for finding a partition of the domain. In Section 4 we compile the details of the algorithm and in Section 5 we shortly summarize our computational experience.

2. BASIC CONSIDERATIONS ABOUT NUMERICAL INVERSION

2.1 Floating Point Arithmetic and Exact Random Variate Generation

In his book Devroye [1986] assumes that *"our computer can store and manipulate real numbers"* (Assumption 1, p. 1). However, in his model "exact random variate generation by inversion" is only possible if the inverse CDF, $F^{-1}(u)$, is available in closed form (using *"fundamental operations"* that are implemented exactly in our computer, Assumption 3). While these assumptions are fundamental for a mathematically rigorous theory of non-uniform random variate generation, they are far from being realistic for common simulation practice.

Virtually all MC or QMC experiments run on a real-world computer that uses floating point numbers. Usually the *double* format of the IEEE floating point standard is used which takes 64 bits for one number resulting in a machine precision of $\epsilon = 2^{-52} \approx 2.2 \times 10^{-16}$ (see Overton [2001] for a survey on the corresponding IEEE 754 standard). Thus a continuous distribution with a very high and narrow

peak or pole actually becomes a discrete distribution, in particular when the mode or pole is located far from 0. Therefore even for an exact inversion method we cannot avoid round-off errors that are bounded from below by the machine precision.

We therefore think that it is more realistic to call an algorithm *"exact"* if its precision is close to machine precision, i.e., its relative error is not much larger than 10^{-15}. To be able to use this idea we must thus start with a definition of *error* and a feasible upper bound for the maximal tolerated deviation.

2.2 Approximation Error and u-Resolution

Let F_a^{-1} denote the approximate inverse CDF. Then we define the absolute *x-error* at some point $u \in (0,1)$ by

$$\varepsilon_x(u) = |F^{-1}(u) - F_a^{-1}(u)| \,.$$

However, it requires the close to exact computation of the inverse CDF and thus it could only be applied for testing an inversion algorithm on a small set of test distributions, for which the inverse CDF is computable. Moreover, a bound for $\varepsilon_x(u)$ requires that an algorithm is very accurate in the far tails of the distributions, i.e., for large values of $F^{-1}(u)$. On the other hand, if we replace the absolute error by the *relative* x-error, $\varepsilon_x(u)/|F^{-1}(u)|$ our algorithm must be very accurate near 0.

A better choice is the *u-error* at a point $u \in (0,1)$ given by

$$\varepsilon_u(u) = |u - F(F_a^{-1}(u))| \,. \tag{1}$$

It has some properties that make it a convenient and practical relevant measure of error in the framework of numerical inversion.

—$\varepsilon_u(u)$ can easily be computed provided that we can compute F sufficiently accurately. Thus the maximal u-error can be estimated during the setup.

—Uniform pseudo-random number generators work with integer arithmetic and return points on a grid. Thus these pseudo-random points have a limited resolution, typically $2^{-32} \approx 2.3 \times 10^{-10}$ or (less frequently) machine precision $2^{-52} \approx 2.2 \times 10^{-16}$. Consequently, the positions of pseudo-random numbers U are not random at all at a scale that is not much larger than their resolution. u-errors can be seen as minor deviations of the underlying uniform pseudo-random points U_i from their "correct" positions. We consider this deviation as negligible if it is (much) smaller than the resolution of the pseudo-random variate generator.

—The same holds for QMC experiments where the F-discrepancy [Tuffin 1997] of a point set $\{X_i\}$ is computed as discrepancy of the set $\{F(X_i)\}$. If the X_i are generated by exact inversion their F-discrepancy coincides with the discrepancy of the underlying low-discrepancy set. Thus $\varepsilon_u(u)$ can be used to estimate the maximal change of the F-discrepancy compared to the "exact" points.

—Consider a sequence of approximations F_n^{-1} to the inverse CDF F^{-1} such that $\varepsilon_{u,n}(u) < \frac{1}{n}$ and let F_n be the corresponding CDF. Then $|F(x) - F_n(x)| = |F(F_n^{-1}(u)) - F_n(F_n^{-1}(u))| = |F(F_n^{-1}(u)) - u| = \varepsilon_{u,n}(u) \to 0$ for $n \to \infty$. That is, the CDFs F_n converge weakly to the CDF F of the target distribution and the corresponding random variates $X_n = F_n^{-1}(U)$ converge in distribution to the target distribution [Billingsley 1986].

We are therefore convinced that the u-error is a natural concept for the approximation error of numerical inversion. We use the maximal u-error as our criterion for approximation errors when calculating inverse CDFs numerically. We call the maximal tolerated u-error of an algorithm the *u-resolution* of the algorithm, denoted by ε_u. In the sequel we consider it as a control parameter for a numerical inversion algorithm. It is a design goal of our algorithm to have

$$\sup_{u\in(0,1)} |u - F(F_a^{-1}(u))| \le \varepsilon_u \ . \tag{2}$$

We should also mention two possible drawbacks of the concept of u-resolution. First, it does not work for continuous distributions with high and narrow peaks or poles. Due to the limitations of floating point arithmetic the u-error is at least of the order of the probability of the mass points described in Sect. 2.1 above. However, this just illustrates the severe problems that floating point arithmetic has with such distributions. Secondly, the simple formula for the x-error

$$\varepsilon_x(u) = \varepsilon_u(u)/f(F^{-1}(u)) + O(\varepsilon_u(u)^2)$$

implies that the x-error of the approximation may be large in the tails of the target distribution. However, we are not considering the problem of calculating exact quantiles in the extremely far tails here. This is (in our opinion) not necessary for the inversion method as the extremely far tails do not influence the u-error.

2.3 Design of an Automatic Inversion Algorithm

The aim of this paper is to develop an inversion algorithm that can be used to sample from a variety of different distributions. The user only has to provide

—a function that evaluates the PDF of the target distribution,
—a "typical point" of the distribution, that is, a point in the domain of the distribution not too far away from the mode, and
—the desired u-resolution ε_u.

We call such algorithms "automatic" or "black box", see Hörmann et al. [2004]. The algorithm uses tables of interpolating polynomials and consists of two parts: (1) the *setup* where all necessary constants for the given distribution are computed and stored in tables; and (2) the *sampling* part where the interpolating polynomials are evaluated for a particular value $u \in (0, 1)$. The setup is the crucial part and we may say that the setup "designs" the algorithm automatically.

The choice of the u-resolution depends on the particular application and is obviously limited from below by the machine precision. It is important that the maximum u-error can be estimated and controlled in the setup, that is, $\varepsilon_u(u)$ must not exceed the requested u-resolution. Moreover, we wish that a u-resolution close to machine precision (say down to 10^{-13}) can be reached with medium-sized tables storing less than 10^4 double constants. We also hope that the sampling part of the algorithm is (very) fast, about as fast as generating exponential random variates by inversion.

It should be clear that for reaching such high aims we also have to accept downsides. For an accurate algorithm we have to accept a slow setup. Of course, we also cannot expect that an automatic algorithm works for all continuous distributions.

For our numerical inversion algorithm we have to assume that the density of the distribution is bounded, sufficiently smooth and positive on its relevant domain. The necessary mathematical conditions for the smoothness can be seen from the error bounds for numerical integration and interpolation; a density with bounded eighth derivative is certainly smooth enough. Moreover, if the density is multimodal, then it must not vanish between its modes. (In practice this means that the density must not be close to zero in a region around a local minimum.) If there are isolated points of discontinuity or where other assumptions do not hold, it is often possible to decompose the domain of the distribution. We thus obtain intervals with smooth PDF and can apply the approximation procedure.

Of course we have to assume that the density function $f(x)$ can be evaluated with an accuracy close to machine precision. Otherwise, if the density is not exactly known then also the rejection method cannot be used to generate random variates from the exact distribution. Depending on the type of the error-bound known for the density evaluation, it may be possible to obtain loose bounds for the resulting maximal u-error. In the sequel we assume that the density can be evaluated up to machine precision and errors from evaluating the density can thus be considered as rounding errors.

2.4 A Short Survey on Numerical Inversion of the CDF

For many years the computation of quantile points of a density has been a challenging and important issue. Many researchers have contributed and suggested carefully crafted algorithms for this task. With the availability of digital computers the requisites for such methods have changed with time. Algorithms that compute the requested quantiles instantly with appropriate accuracy now replace rough rule-of-thumb estimations or precomputed tables. Increasing computer power as well as demand for more accurate values was the incitement for the development of many sophisticated algorithms. The short summary of Thomas et al. [2007] for the Gaussian distribution reflects these efforts.

Kennedy and Gentle [1980, Chapter 5] note that *"reviewing this area of statistical computing is difficult because a multitude of different numerical methods are employed, often in different forms, to construct algorithm."* They have identified the following general methods and techniques:

—Exact relationships between distributions
(e.g., between the F and the beta distribution).

—Approximate transformations
(e.g., Wilson and Hilferty [1931] for χ^2 distributions).

—general series expansions
(e.g., Taylor series or the Cornish-Fisher expansion [Cornish and Fisher 1937; Fisher and Cornish 1960]).

—Closed form approximations with polynomials and rational functions.

—Continued fractions.

—Gaussian and Newton-Cotes quadrature for computing the CDF, and

—numerical root finding: Newton's method, secant method, interval bisectioning.

These technique are also combined in several ways. E.g., Hastings [1955] estimates the inverse Gaussian CDF $x(u)$ as a rational function of $\sqrt{-\log u^2}$. Wichura [1988] uses different transformations in the central part and the tails of the normal distribution. In statistical software (e.g., R [R Development Core Team 2008]) the result of such approximations are often post-processed using Newton's method or bisectioning to obtain an accuracy close to machine precision. This is in particular necessary in the case of distributions with parameters. It is of course beyond the scope of this paper to describe any of the neat algorithms that have been developed during the last five decades. We refer to the book of Kennedy and Gentle [1980] for a survey of the general methods and of specialized algorithms for the most important distributions. Johnson et al. [1994; 1995] describe many approximations for all common distributions and give an extensive literature.

For the development of fast universal algorithms most of these methods are not applicable. Either they are too slow (numerical root finding) or they require transformations that have to be especially tailored for the target distribution. Series expansion require informations that are hard to obtain for non-common distributions, e.g., higher order derivatives or cumulants of the distribution. In contrast piecewise polynomial functions are well suited for the approximation of the inverse CDF. Ahrens and Kohrt [1981], Ulrich and Watson [1987] and Hörmann and Leydold [2003] propose algorithms that are based on this approach. The last paper assumes that the CDF of the distribution is available while the first two papers only require the density and use some integration routines. Short descriptions of these methods are postponed to Sect. 6 where we also provide a comparison with the new algorithm.

It is known that rational functions would be superior to polynomial approximation of inverse CDFs. However, Monahan [2001, Sect. 7.2, p. 142] remarks that *"mathematically [rational functions] are not as easy to work with as polynomials. [...] Some drawbacks need to be faced, since (i) the system of equations may not yield a unique solution, (ii) a unique solution is necessary but not sufficient for determining the rational function interpolated, and (iii) the solution to the system of equations is often badly conditioned."* Furthermore, it is hard to design a method that estimates the approximation error automatically. The same also holds for continued fraction. (Rational expressions can be converted into continued fractions by Viskovatov's method.)

Ulrich and Watson [1987] suggest another method. Inverse CDFs can be seen as solutions of the differential equation $x'(u) = 1/f(x(u))$ and some starting value $x(u_0) = x_0$. Thus any ODE solver could be used for computing the inverse CDF numerically. Leobacher and Pillichshammer [2002] utilized this approach for the hyperbolic distribution. However, they had to treat the tails separately as the ODE cannot be solved near the points $u = 0$ and $u = 1$ numerically. Ulrich and Watson [1987] developed fourth and fifth order Runge-Kutta type approximations to the inverse CDF.

3. BUILDING BLOCKS OF NUMERICAL INVERSION

As we start with the density $f(x)$ we have to solve the following problems:

(1) *Computational domain:* Find the computationally relevant domain $[b_l, b_r]$ of the

distribution, that is, the region where the construction of our approximation of the inverse CDF is numerically stable, and where the probability of falling into this region is sufficiently close to 1.

(2) *Subintervals:* Divide the domain of the distribution into intervals $[a_{i-1}, a_i]$, $i = 1, \dots k$, with $b_l = a_0 < a_1 < \ldots < a_k = b_r$.

(3) *Interpolation:* For each interval we approximate the inverse CDF F^{-1} on the interval $[F(a_{i-1}), F(a_i)]$ by interpolating the construction points $(F(x_j), x_j)$ for some points $a_{i-1} = x_0 < x_1 < \ldots x_n = a_i$. Notice that it is never necessary to evaluate $F^{-1}(u)$.

(4) *Numerical integration:* Compute the approximate CDF, $F_a(x)$, by iteratively computing $\int_{x_{j-1}}^{x_j} f(x)\,dx$ for $j = 1, \dots, n$ on each interval.

For Task (4) we use (adaptive) Gauss-Lobatto quadrature. For Task (3) we found that Newton's recursion for the interpolating polynomial ("Newton's interpolation formula") with a fixed number of points is well-suited. Both, numerical integration and interpolation lead to small errors when applied on short intervals and they allow the estimation of u-errors on each interval. Thus we can accomplish Task (2) by selecting proper intervals which are short enough to gain sufficient accuracy but not too short thus avoiding needless large tables. Notice that by this approach the computation of the approximation is carried out on each interval independently from the others. We will see that using the same intervals for integration and for interpolation leads to significant synergies. Task (1) is important as our approach only works for distributions with bounded domains. Moreover, regions where the CDF is extremely flat (as in the tails of the distributions) result in an extremely steep inverse CDF and its polynomial interpolation becomes numerically unstable.

The sampling part of the algorithm is straightforward. We use indexed search [Chen and Asau 1974] to find the correct interval together with evaluation of the interpolation polynomial.

3.1 Newton's Interpolation Formula

Polynomial interpolation for approximating a function $g(x)$ on some interval $[x_0, x_n]$ is based on the idea to use a polynomial $P_n(x)$ of order n such that

$$g(x_i) = P_n(x_i), \quad \text{for } i = 0, \dots, n,$$

where $x_0 < x_1 < \ldots < x_n$ are some fixed points. Note, that we use the borders of the interval, x_0 and x_n, as interpolation points to avoid discontinuities of the approximate polynomials at successive intervals. For smooth functions the approximation error at a point $x \in (x_0, x_n)$ is given by

$$|g(x) - P_n(x)| = \frac{|g^{(n+1)}(\xi)|}{(n+1)!} \prod_{i=0}^{n} |x - x_i| \quad \text{for some } \xi \in (x_0, x_n). \tag{3}$$

Thus for a function with bounded $(n+1)$-st derivative the approximation error is of order $O((x_n - x_0)^{n+1})$ and can be made arbitrarily small by using short intervals.

Newton's interpolation formula uses a numerically stable representation of the polynomial $P_n(x)$ and a simple and elegant recursion to calculate the coefficients of that representation. Using the ansatz (see, e.g., Schwarz and Klöckler [2009,

Sect. 3.1] or Dahlquist and Björck [2008, Sect. 4.2.1])

$$P_n(x) = c_0 + \sum_{k=1}^{n} c_k \prod_{i=0}^{k-1} (x - x_i)$$

one finds that

$$c_k = g[x_0, x_1, \ldots, x_k] \qquad \text{for } k = 0, \ldots, n,$$

where the *divided differences* are recursively defined by

$$g[x_0, x_1, \ldots, x_k] = \frac{g[x_1, \ldots, x_k] - g[x_0, \ldots, x_{k-1}]}{x_k - x_0} \quad \text{and} \quad g[x_i] = g(x_i).$$

This formula allows to compute the coefficients c_k using Routine 1. The polynomial can be evaluated at some point $x \in [x_0, x_n]$ using the Horner like scheme in Routine 2.

Routine 1 `NCoef` (Newton-Coefficients)

Input: Nodes $x_0 < \ldots < x_n$, values $g(x_0), \ldots, g(x_n)$.
Output: Coefficients $c_0, \ldots, c_n$ for interpolating polynomial P_n.

```
1: for i = 0, ..., n do
2:     c_i ← g(x_i).
3: for k = 1, ..., n do
4:     for i = n, n − 1, ..., k do
5:         c_i ← (c_i − c_{i−1})/(x_i − x_{i−k}).
6: return c_0, ..., c_n.
```

Routine 2 `NEval` (Newton-Evaluate)

Input: Coefficients c_k of P_n, nodes $x_0, \ldots, x_n$, point $x \in [x_0, x_n]$.
Output: Value of $P_n(x)$.

```
1: p ← c_n.
2: for k = n − 1, n − 2, ..., 0 do
3:     p ← c_k + (x − x_k) p.
4: return p.
```

We used *Chebyshev points* of $(n+1)$-st order as nodes for P_n, i.e., the roots of the Chebyshev polynomial of the first kind of order $n+1$, T_{n+1}, given by

$$\cos\left(\frac{2k+1}{n+1} \cdot \frac{\pi}{2}\right), \quad \text{for } k = 0, 1, \ldots, n,$$

rescaled to our intervals $[x_0, x_n]$ such that the smallest and the largest root are mapped on the boundary points x_0 and x_n. These points lead to a minimal upper bound for the maximal approximation error. We call these points the *rescaled Chebyshev points* in the following. For the interval $[0, 1]$ we find

$$x_k = \frac{\sin(k\,\phi)\,\sin((k+1)\,\phi)}{\cos(\phi)} \quad \text{for } k = 0, 1, \ldots, n, \quad \text{where } \phi = \frac{1}{(n+1)}\,\frac{\pi}{2}\,. \qquad (4)$$

3.2 Inverse Interpolation

For an interval $[F(a_{k-1}), F(a_k)]$ we apply Newton's interpolation formula, to construct an approximating polynomial F_a^{-1} for the inverse CDF F^{-1}. For the nodes we use $u_i = F(x_i)$ where x_i are the rescaled Chebyshev points on the interval $[a_{k-1}, a_k]$. The values of F^{-1} at the nodes are $F^{-1}(u_i) = x_i$. This approach avoids the evaluation of the inverse CDF. Moreover, it leads to close to minimal approximation errors where F is nearly linear within an interval $[a_{k-1}, a_k]$ (which is often the case for the center of the distribution). It may lead to larger than optimal errors in other parts, in particular in the tails, but it is still better than using equidistant u_i. A disadvantage is that we have to store the values of u_i in a table (see also Remark 3 below). We finally decided to use this simple approach as it leads to a stable setup.

The estimation of the interpolation error is a crucial part of an automatic algorithm. As direct consequence of (3) we find for the x-error

$$\varepsilon_x(u) = |F^{-1}(u) - F_a^{-1}(u)| = \frac{|(F^{-1})^{(n+1)}(\zeta)|}{(n+1)!} \prod_{i=0}^{n} |u - u_i| \tag{5}$$

where $(F^{-1})^{(n+1)}(\zeta) = \left(\frac{d}{dv}\right)^{n+1} F^{-1}(v)\big|_{v=\zeta}$ denotes the $n+1$st derivate of F^{-1} at some point $\zeta \in [F(a_{k-1}), F(a_k)]$. Hence we obtain an upper bound for the u-error of the approximation.

LEMMA 1. *Let F_a^{-1} be a polynomial of order n, approximating F^{-1} over $[F(a_{k-1}), F(a_k)]$ for some interval $[a_{k-1}, a_k]$, as described above. Assume that F^{-1} is $n+1$-times continuously differentiable. If F^{-1} and F_a^{-1} have the same image, then*

$$\varepsilon_u(u) \le \max_{\substack{x\in[a_{k-1},a_k] \\ v\in[F(a_{k-1}),F(a_k)]}} f(x)\, \frac{|(F^{-1})^{(n+1)}(v)|}{(n+1)!} \prod_{i=0}^{n} |u - u_i| \,. \tag{6}$$

Notice that the last condition is satisfied when F_a^{-1} is monotone, since F^{-1} and F_a^{-1} coincide on $F(a_{k-1})$ and $F(a_k)$.

PROOF. Let $u \in [F(a_{k-1}), F(a_k)]$ and $u_a = F(F_a^{-1}(u))$. Then $\varepsilon_u(u) = |u - u_a|$ and $\varepsilon_x(u) = |F^{-1}(u) - F^{-1}(u_a)|$. Hence by the mean value theorem there exists a $\zeta \in (u, u_a)$ such that $\varepsilon_x(u)/\varepsilon_u(u) = (F^{-1})'(\zeta) = 1/F'(\xi) = 1/f(\xi)$ where $\xi = F^{-1}(\zeta)$. As F^{-1} and F_a^{-1} have the same image, we find $\xi \in [a_{k-1}, a_k]$ and thus $\varepsilon_u(u) \le \max_{x\in[a_{k-1},a_k]} f(x)\, \varepsilon_x(u)$. Hence the result follows from (5). □

It is possible to use this bound to derive rigorous error bounds on the approximation error as the derivatives of F^{-1} can be calculated using only derivatives of $f(x)$ by means of Faá di Bruno's formula (see Sect. 3.7 below for an example). However, computing and implementing higher order derivatives is quite tedious for all and virtually impossible for many densities. Thus it is not a convenient ingredient of a black-box algorithm.

We therefore also consider, how to estimate the u-error without using the above bound. Instead it is useful to consider the approximation

$$\varepsilon_u(u) \approx f(F^{-1}(u))\, \varepsilon_x(u) \approx f(F^{-1}(u))\, \frac{|(F^{-1})^{(n+1)}(u)|}{(n+1)!} \prod_{i=0}^{n} |u - u_i| \,.$$

For many distributions we made the following observation: For (very) short intervals, compared to $\prod_{i=0}^{n}(u-u_i)$, the term $f(F^{-1}(u))\,(F^{-1})^{(n+1)}(u)$ has only little influence on $\varepsilon_u(u)$. (If this is not the case, then the interpolation error is large and we have to use a shorter interval anyway.) Thus the maximal u-error in a short interval is attained very close to the extrema of $\prod_{i=0}^{n}|u-u_i|$ that we denote by t_i, $i=1,\ldots,n$. We can therefore use

$$u\text{-error} \approx \max_{i=1,\ldots,n} |t_i - F_i(F_a^{-1}(t_i))| \tag{7}$$

as a simple estimate for the u-error of the interpolation. Here F_i denotes the approximate CDF computed using numerical integration of the PDF.

For the polynomial $p(u)=\prod_{i=0}^{n}(u-u_i)$ we find

$$p'(u) = \sum_{k=0}^{n}\prod_{\substack{i=0\\ i\neq k}}^{n}(u-u_i) = p(u)\sum_{k=0}^{n}\frac{1}{u-u_k}\ .$$

Since the points $u_0<\ldots<u_n$ are distinct, $p'(u)$ cannot be zero at u_k, $k=0,\ldots n$. Thus we only need to find the roots of $\sum_{k=0}^{n}\frac{1}{u-u_k}$ (approximately). We found out that this can be done efficiently using only two iterations of Newton's root finding method, where starting with $\hat{t_i}=(u_{i-1}+u_i)/2$ we use the recursion

$$\hat{t}_{i,new} = \hat{t}_i + \frac{\sum_{k=0}^{n}(1/(\hat{t}_i-u_k))}{\sum_{k=0}^{n}(1/(\hat{t}_i-u_k)^2)}\ .$$

The entire algorithm for computing the test points t_i is compiled in Routine 3.

Routine 3 `NTest` (Newton-Testpoints)

Input: Nodes $u_0<\ldots<u_n$.
Output: Test points $t_1<\ldots<t_n$.

```
1: for i = 1,...,n do
2:    t_i ← (u_{i-1} + u_i)/2.
3:    for j = 1,2 do                                  ▷ 2 Newton steps
4:       s ← 0,    sq ← 0.
5:       for k = 0,...,n do
6:          s ← s + 1/(t_i − u_k),    sq ← sq + 1/(t_i − u_k)^2.
7:       t_i ← t_i + s/sq.
8: return t_1,...,t_n.
```

Heuristic (7) works for quite a large class of simulation problems. However, although based on some mathematical arguments there is no guarantee that it works reliably for every distribution.

Notice that the computation of the first derivative of $\varepsilon_u(u)$ only requires one evaluation of the density and one of the derivative of the interpolating polynomial. The derivative can be calculated by inserting only two additional statements into Routine 2: $p' \leftarrow 0$ (initialize p') before the loop is started and $p' \leftarrow p + (t-x_k)\,p'$ (apply the product rule) as first statement within the loop. The resulting modified Routine 2 returns the value p of the interpolating polynomial at x and its derivative p', as well. As the error is known to be zero at u_i we can use the very stable bisection

method with starting interval $[u_{i-1}, u_i]$ to find the root of the derivative of $\varepsilon_u(u)$ and thus the local maximum in that region. This method works reliably as long as this maximum is unique in each of the subintervals $[u_{i-1}, u_i]$. Of course the numerical search for the n local extrema of the u-error requires a lot of evaluations of the density and thus slows down the setup considerably. It is interesting that in most cases this method leads to the same decomposition of the domain into intervals $[a_{k-1}, a_k]$ as evaluating the u-error only at the testpoints.

Note that it is not easy to decide between the two variants: numerical search for each local maximum of the u-error, or evaluation of the u-error only at the test points. It is a tradeoff between setup time and (more) reliable error control. As our empirical results for the simple error control were always satisfying we decided to use it as default method for our algorithm.

Monotonicity of the approximated inverse CDF, F_a^{-1}, might also be an issue for some applications. There exists a lot of literature on shape-preserving (mostly cubic) spline interpolation. However, methods for constructing monotone polynomials that interpolate monotone points are rather complicated (e.g., Costantini [1986]). Thus we have decided to use a rather simple technique and only check the monotonicity at the points $F_a^{-1}(t_i)$ that we already use in (7) for estimating the u-error. If monotonicity is of major importance for an application, it is possible to check for monotonicity by evaluating the first derivative of the approximating polynomial (which is easy as mentioned above) at many points, or by searching numerically for negative values of that derivative.

To increase the numeric stability of the interpolation of the inverse CDF we use a transformed version F_k of the CDF on each interval $[a_{k-1}, a_k]$. Therefore we define $F_k(x) = F(x + a_{k-1}) - F(a_{k-1}) = \int_{a_{k-1}}^{a_{k-1}+x} f(t)\,dt$ for $x \in [0, a_k - a_{k-1}]$. Then the problem of inverting $F(x)$ on $[a_{k-1}, a_k]$ is equivalent to inverting $F_k(x)$ on $[0, a_k - a_{k-1}]$: for a $u \in [F(a_{k-1}), F(a_k)]$ we get $F^{-1}(u)$ by

$$F^{-1}(u) = a_{k-1} + F_k^{-1}(u - F(a_{k-1}))\ .$$

This transformation has two advantages. The computation of $F_k(x)$ by integrating $\int_{a_{k-1}}^{a_{k-1}+x} f(t)\,dt$ is numerically more stable because the subtraction $F(x + a_{k-1}) - F(a_{k-1})$, that might cause loss of accuracy, is no longer necessary. This improves in particular the numeric precision in the right tail of the distribution. Moreover, the first node of our interpolation problem is always $(0, 0)$ which saves memory and computations.

Remark 2. We use linear interpolation as a fallback when Newton's interpolation formula fails due to numerical errors.

Remark 3. An alternative approach is to use the optimal interpolation points, i.e., the Chebyshev points rescaled for the interval $[F(a_{k-1}), F(a_k)]$. This is convenient as, up to a linear transformation, they are the same for all intervals and we need not store them in a table. Moreover, this also holds for the test points t_i for the error estimate (7) and, therefore, we would not need Routine `NTest`. However, the main drawback of this alternative is that we have to invert the CDF to calculate $x_i = F^{-1}(u_i)$ which may lead to numerical problems due to rounding errors especially in the tails.

Remark 4. We also made experiments with other interpolations. However, we have decided against these alternatives. Hermite interpolation (see [Hörmann and Leydold 2003]) has slightly larger u-errors for a given order of polynomial and requires (higher order) derivatives of the density. Moreover, it was numerically less stable in regions with flat densities, especially in the tails of the distribution. On the other hand, continued fractions have the advantage that we can extrapolate towards poles but we were not able to estimate interpolation errors during the setup.

3.3 Gauss-Lobatto Quadrature

There exist many different quadrature rules. We are interested in very precise results for smooth f on short intervals. In our experiments with several standard distributions we noticed that Gauss-Lobatto quadrature with 5 nodes resulted in very small errors for all intervals that we obtained from the interpolation algorithm. For the interval $[0,1]$ it uses the nodes $c_1 = 0$, $c_2 = \frac{1}{2} - \sqrt{3/28}$, $c_3 = \frac{1}{2}$, $c_4 = \frac{1}{2} + \sqrt{3/28}$, and $c_5 = 1$ with corresponding weights $w_1 = w_5 = \frac{9}{180}$, $w_2 = w_4 = \frac{49}{180}$, and $w_3 = \frac{64}{180}$. Then for a density f on $[a, a+h]$ we have

$$\int_a^{a+h} f(x)\,dx \approx \hat{I}_{(a,a+h)}[f] = \sum_{i=1}^{5} w_i\, h\, f(a + c_i\, h)\ . \tag{8}$$

The integration error for an eight times continuously differentiable density f is given by [Abramowitz and Stegun 1972]

$$\left| \int_a^{a+h} f(x)\,dx - \hat{I}_{(a,a+h)}[f] \right| \le 7.03 \cdot 10^{-10}\, h^9 \max_{\xi \in [a,a+h]} |f^{(8)}(\xi)|\ . \tag{9}$$

Thus numerical integration works for smooth $f(x)$ with bounded 8th derivative.

Remark 5. We found that errors for Gauss-Lobatto quadrature are usually much smaller than those of interpolation. Note, however, that (e.g.) for the Gamma distribution with shape parameter $\alpha < 2$ we have unbounded derivatives at zero. This leads to an integration error for the interval $(0, h)$ which increases linearly with h. But this difficulty can be overcome by using adaptive integration.

Adaptive integration is a standard method for numerically integrating $f(x)$ on long intervals. In addition, it allows to (roughly) estimate the integration error without considering mathematical bounds. We use a variant where we halve the intervals, until the total integral does not change any more, see Routine 4 (`AGL`).

In practice evaluations of f and the two subintegrals for I_1 are passed to `AGL` in each recursion step. Thus a single call to this routine requires 11 evaluations of f, and each additional step 6 evaluations. We also need a rough estimate for $\int_{b_l}^{b_r} f(x)\,dx$ in order to set tolerance tol as the function f can be any positive multiple of a density function.

Notice that $|I_0 - I_1|$ is just an estimate for the integration error and does not provide any upper bound. Many papers suggest to replace tol by $tol/2$ when recursively calling the adaptive quadrature algorithm in order to obtain an upper bound for the estimated error. However, we observed that halving the tolerance

Routine 4 `AGL` (Adaptive-Gauss-Lobatto)

Input: Density $f(x)$, domain $[a, a+h]$, tolerance tol.
Output: $\int_a^{a+h} f(x)\,dx$ with estimated maximal error less than tol.

```
1: I_0 ← Î_(a,a+h)[f].
2: I_1 ← Î_(a,a+h/2)[f] + Î_(a+h/2,a+h)[f].
3: if |I_0 − I_1| < tol then
4:    return I_1.
5: else
6:    return (AGL(f, (a, a + h/2), tol) + AGL(f, (a + h/2, a + h), tol)).
```

leads to severe problems for distributions with heavy tails (e.g., the Cauchy distribution) where it never reaches the precision goal. Thus we followed the version of adaptive integration described by Gander and Gautschi [2000]. They argue[1] that halving the tolerance for every subdivision leads to lots of unnecessary function evaluations without providing an upper bound for the (true) integration error, that is not available with that type of error bound anyway. In all our experiments, the errors when using Routine 4 were smaller than required. We have even observed in our experiments that for nearly all cases the intervals for Newton's interpolation formula are adequately short for simple (non-adaptive) Gauss-Lobatto quadrature (8) to obtain sufficient accuracy. However, for distributions with high tails (e.g., Cauchy) or unbounded derivatives (e.g., Gamma with shape parameter α close to one) we needed adaptive integration. The following procedure was quite efficient then:

0. Roughly compute $I_0 = \hat{I}_{(b_l,b_r)}[f]$ to get maximal tolerated error tol.
1. Compute $\hat{I}_{(b_l,b_r)}[f]$ with required accuracy by adaptive Gauss-Lobatto quadrature using Routine `AGL` and store subinterval boundaries and CDF values. We used $tol = 0.05\, I_0\, \varepsilon_u$.
2. When integrals $\hat{I}_{(x_{j-1},x_j)}[f]$ have to be computed we use the intervals and the CDF values from Step 1 above together with simple Gauss-Lobatto quadrature.

Remark 6. There exist many other quadrature rules as well. Some of these may be a suitable replacement. However, the arguments for our procedure are as following:

—Adaptive integration provides flexibility. It works for (sufficiently smooth) densities without further interventions.

—Using the same quadrature rule for each recursion of adaptive integration as well as for the simple quadrature rule allows to store and reuse intervals and CDF values. Thus we have sufficient accuracy with simple quadrature on arbitrary subintervals.

—The boundary points of the intervals should be used as nodes of adaptive quadrature. Thus they can be reused for further recursion steps.

[1] Private communication from Walter Gander.

—The number of nodes for the (simple) quadrature rule should in most cases provide sufficient accuracy for the integrals between the construction points of the interpolating polynomials; but not more to avoid futile density evaluations.

For these reasons we have rejected Gauss-Legendre quadrature with 4 nodes. It has an integration error similar to Gauss-Lobatto quadrature with 5 nodes but as the latter uses the interval endpoints as nodes it is especially suited for recursive subdivisions of the intervals. Then it requires only 6 additional evaluations of f in opposition to Gauss-Legendre where 8 evaluations are necessary. Moreover for Gauss-Lobatto quadrature three of the 5 nodes and all weights are rational numbers and can be stored without rounding errors.

3.4 Cut-off Points for the Computational Domain

Newton's interpolation formula becomes numerically unstable when the inverse CDF is very steep. Thus numerical problems arise for the case where the density f is close to zero over some subinterval. This leads to cancellation errors and overflows when calculating the coefficients of the interpolating polynomial. For many (in particular for all unimodal) distributions this can occur only in the tails. It is therefore important for the successful application of our algorithm that we cut off these parts from the domain of a distribution. Of course the tail regions must have small probabilities as they contribute to the total u-error. We used a probability of $0.05\,\varepsilon_u$ for either tail.

For this task we have to find easy-to-compute approximations or bounds for the quantiles in the far tails. For log-concave densities it is easy to get such a bound by replacing the tail by an appropriate exponential tail. To get a generally applicable method we consider the more general concept of T_c-concave densities [Hörmann et al. 2004, Sect. 4.3]. Let T_c be the strictly monotone increasing transformation $T_c(x) = \operatorname{sgn}(c)\, x^c$ if $c \neq 0$, and $T_0(x) = \log(x)$ otherwise. We call a distribution T_c-concave if its transformed density $\tilde{f}(x) = T_c(f(x))$ is concave for a fixed value of c (log-concave is then the special case $c = 0$). Now we consider the tangent $\tilde{g}(x) = \tilde{f}(p) + \tilde{f}'(p)\,(x-p)$ on $\tilde{f}(x)$ in a point p of the left tail region with c selected such that $\tilde{f}(x)$ is concave. Let us denote by $g(x)$ the function which one obtains by applying the inverse transformation T_c^{-1} on $\tilde{g}(x)$,

$$g(x) = T_c^{-1}(\tilde{g}(x)) = \begin{cases} f(p)\,\left(1 + c\,\frac{f'(p)}{f(p)}\,(x-p)\right)^{1/c} & \text{for } c \neq 0\,, \\ f(p)\,\exp\left(\frac{f'(p)}{f(p)}\,(x-p)\right) & \text{for } c = 0\,. \end{cases}$$

The function $G(x) = \int_{-\infty}^{x} g(t)\,dt$ or $G(x) = \int_{b_0}^{x} g(t)\,dt$ (if the domain of g is bounded at b_0) can be computed and inverted in closed form and thus it is no problem to solve the equation $G(p) = \varepsilon = 0.05\,\varepsilon_u$. Denoting its root by p^* we get the simple formulas

$$p^* = \begin{cases} p + \frac{f(p)}{c\,f'(p)}\left(\left(\frac{\varepsilon\,|f'(p)|\,(1+c)}{f(p)^2}\right)^{c/(1+c)} - 1\right) & \text{for } c \neq 0, \\ p + \frac{f(p)}{f'(p)}\,\log\frac{\varepsilon\,|f'(p)|}{f(p)^2} & \text{for } c = 0. \end{cases} \tag{10}$$

Note that by using the absolute value $|f'(p)|$ we obtain for both $c \neq 0$ and $c = 0$

a single formula applicable for both, the right and the left tail. The result p^* can be used as new value for p thus defining a simple recursion that converges very fast close to exact results.

We have assumed above that c was selected such that f is T_c-concave (i.e. $\tilde{f}(x)$ is concave). Then the function $g(x)$ is an upper bound for $f(x)$. This clearly implies that p^* is always a lower bound for the quantile in the left tail and an upper bound for the quantile in the right tail. Thus the resulting value p^* of the described method is guaranteed to cut off less than the desired probability $0.05\,\varepsilon_u$ if the required constant c is known. This is the case, e.g., for the Normal, beta and gamma distributions that are all log-concave ($c = 0$). For the t-distribution with ν degrees of freedom it is easy to show that it is T_c-concave for $c = -1/(1+\nu)$. We tested the cut-off procedure for all these distributions and obtained very close bounds for the required quantiles. To be precise the cut-off procedure above does not require that the density is T_c-concave everywhere, it is enough that it is T_c-concave in the tail.

Of course it is possible that the value of c that leads to a T_c-concave tail is not known. Fortunately, this situation can be solved by means of the concept of *local concavity* of a density f at a point x [Hörmann et al. 2004, Sect. 4.3]. The local concavity is the maximal value for c such that the transformed density $\tilde{f}(x) = T_c(f(x))$ is concave near x. For a twice differentiable density f it is given by

$$\mathrm{lc}_f(x) = 1 - \frac{f''(x)\,f(x)}{f'(x)^2}$$

and can be calculated sufficiently accurately by using

$$\mathrm{lc}_f(x) = \lim_{\delta\to 0}\left(\frac{f(x+\delta)}{f(x+\delta)-f(x)} + \frac{f(x-\delta)}{f(x-\delta)-f(x)}\right) - 1\,.$$

Taylor series expansion reveals that the approximation error is $O(\delta^2)$ if f is four times continuously differentiable.

Notice that in the left tail $\tilde{f}(x)$ is concave for $x \le p$ whenever $\mathrm{lc}_f(x) \ge c$ for all $x \le p$. If we assume that $\mathrm{lc}_f(x)$ is approximately constant in the far tails we can use the recursion (10) with $c = \mathrm{lc}_f(x)$. Of course the resulting cut-off value is no longer guaranteed to be the bound of the required quantile as we do not have an assumption on the tail behavior of f. Still that approach works correctly and is stable for the distributions we tried.

Remark 7. There is no need for a cut-off point when the domain is bounded from the left by b_l, and $f(b_l)$ or $f'(b_l)$ is greater than zero. Analogously for a right boundary b_r.

3.5 Construct Subintervals for Piecewise Interpolation

We have to subdivide the domain of the distribution into intervals $[a_{i-1}, a_i]$, $i = 1, \ldots k$, with $b_l = a_0 < a_1 < \ldots < a_k = b_r$. We need sufficiently short intervals to reach our accuracy goal. On the other hand many intervals result in large tables. We use a simple technique to solve this problem: We construct the intervals starting from the left boundary point of the domain and proceed to its right boundary. We start with some initial interval length, compute the interpolating polynomial and

estimate the error using (7). If the error is larger than ε_u we have to shorten the interval and try again. Otherwise we fix that interval and store its interpolation coefficients and proceed to the next interval. If the error for the last interval was much smaller than required we try a slightly longer one now.

Remark 8. An alternative approach is interval bisection: Start with a partition of the (computational) domain $[b_l, b_r]$. Whenever the u-error is too large in a interval it is split into two subintervals. This procedure is used in [Hörmann and Leydold 2003] (where the CDF is directly available) but results in a larger number of intervals and thus larger tables. For the setting of this paper, where we start from the PDF and combine numeric integration with interpolation, interval bisection is less suited.

3.6 Adjust Error Bounds

We have several sources of numerical error: Cutting off tails, integration errors and interpolation errors. As we want to control the maximal error we have to adjust the tolerated u-error in each of these steps. Let $\breve{\varepsilon}_u$ denote the requested u-resolution. Then we use $\varepsilon_u = 0.9\,\breve{\varepsilon}_u$ for the maximal tolerated error for the interpolation error as computed in (7). For the probabilities of the truncated tails we use $0.05\,\varepsilon_u$ and for the integration error we allow at most $0.05\,I_0\,\varepsilon_u$. By this strategy the total u-error was always below $\breve{\varepsilon}_u$ for all our test distributions (when $\breve{\varepsilon}_u \geq 10^{-12}$).

We can conclude from bound (6) that the number of intervals will be moderate for interpolation of order n if the density has a bounded n-th derivative (or can be decomposed into intervals with bounded n-th derivative). The number of necessary intervals may get very large if that assumption is not fulfilled but still the interpolation error converges to zero when the interval lengths are tending to zero.

For Gauss-Lobatto integration bound (9) indicates that a bounded eighth derivative is required to obtain fast convergence. Otherwise the number of intervals of our algorithm remains unchanged but a large number of subdivisions and evaluations of the density may be required in the adaptive integration algorithm Routine 4. The integration error tends to zero if the number of subdivisions is increased further and further for any continuous bounded density f.

3.7 Rigorous Bounds

We have described both rigorous mathematical bounds and convenient approximations to assess the maximal u-error of an inversion algorithm. As these approximations lead to precise error estimates in our experiments and are much easier to use, we mainly considered the details of that approach. Still this should not give the wrong impression that it is not possible to calculate and use the exact mathematical bounds given above. As a matter of fact we can calculate the exact error bounds for any density f that is sufficiently smooth (i.e., it has bounded eighth derivative) if we can evaluate the eighth derivative of f. (This is no problem in practice if f can be written in closed form.) We demonstrate here the use of those bounds to develop the inversion algorithm for a prominent example, the standard normal distribution. It works analogously for other distributions provided that the respective CDF and density are "sufficiently nice".

We first have to fix the requested u-resolution $\breve{\varepsilon}_u$ and the order n of the in-

terpolation polynomial, say $n = 5$. It is easy to verify that the density of the normal distribution is log-concave. Thus we can use recursion (10) with $c = 0$ to find appropriate cut-off points. Function $G(p)$ can then be used to verify that these points are respective upper and lower bounds for the corresponding quantiles. It is straightforward to check that for the eighth derivative $f^{(8)}(x) \le f^{(8)}(0) = 105/\sqrt{2\,\pi}$. Thus (9) implies that the integration error is bounded by $2.945 \cdot 10^{-8}\, h^9$, where h denotes the interval length. Assume for simplicity that $0 \le a_{k-1} < a_k$. Then it is easy to see that density f is monotonically decreasing on $[a_{k-1}, a_k]$ whereas $(F^{-1})^{(6)}(F(x)) = 8e^{3x^2}\pi^3 x\left(120x^4 + 326x^2 + 127\right)$ is increasing. Moreover, $|p(u)| = \prod_{i=0}^{5} |u - u_i| \le \frac{1}{6}\left(\frac{5}{6}\right)^5 h^6 \le 0.07\, h^6$. Thus by (6) we find for the interpolation error in $[a_{k-1}, a_k]$

$$\varepsilon_u(u) \le f(a_{k-1})\,(F^{-1})^{(6)}(F(a_k))\frac{0.07}{6!}(F(a_k) - F(a_{k-1}))^6\,.$$

It should be noted here that the bound is quite rough. Compared to using rescaled Chebyshev points for the u_i our pessimistic bound for the maximum of $|p(u)|$ is more than 100 times larger. Still it is of some interest as it demonstrates that we can obtain an exact upper bound for the interpolation error.

Using the above bounds for the integration and the interpolation error we can directly calculate the maximal error for a given selection of the intervals. It is also no problem to find for a given maximal acceptable error ε_u a decomposition into intervals that guarantees that the maximal error is really smaller than ε_u.

Remark 9. We must emphasize here, that the error bound is only valid for precise arithmetic. Therefore the round-off errors of the floating point arithmetic used on real-world computers may still lead to larger than required u-errors. The u-error may even become very large for extreme parameter settings. Consider for example the normal distribution with $\mu = 10^{20} + 50$ and $\sigma = 1$. Due to the floating point arithmetic the returned value is always 10^{20}. This implies that the u-error in that example is always equal to u and thus takes values up to almost 1.

4. THE ALGORITHM

Algorithm `NINIGL` (Numerical Inversion with Newton Interpolation and Gauss-Lobatto integration) collects all building blocks in a lean form.

5. IMPLEMENTATION AND COMPUTATIONAL EXPERIENCE

We coded Algorithm `NINIGL` and added it as new method PINV to our C library UNU.RAN [Leydold and Hörmann 2009b] for random variate generation. Our major concerns were stability and reliability, that is, the algorithm should be able to handle numerically difficult distributions and the maximal u-error should not exceed the maximum tolerated error ε_u given by the user. (Of course we cannot expect that it works for *every* distribution due to limitations of floating point arithmetic.) We used the R Project for Statistical Computing [R Development Core Team 2008] as a convenient environment for doing stochastic simulations. Hence we have prepared package Runuran [Leydold and Hörmann 2009a] to make our UNU.RAN library accessible within R. This allows us to test our algorithms with CDF implementations that are independent from our C code. For moderate (or

Algorithm 1 `NINIGL`

Input: Density $f(x)$, center x_c of distribution, u-resolution ε_u, order n.
Output: Random variate with approximate density f and maximal u-error ε_u.

▷ Adjust

1: $\varepsilon_u \leftarrow 0.9\,\varepsilon_u$.

▷ Preprocessing

2: Find points $\tilde{b}_l < x_c < \tilde{b}_r$ with $f(\tilde{b}_l) \approx f(\tilde{b}_r) \approx 10^{-13} f(x_c)$.
3: Roughly Estimate $I_0 \leftarrow \hat{I}_{(\tilde{b}_l,\tilde{b}_r)}[f]$.
4: Find cut-off points b_l and b_r for computational domain with $\mathrm{Prob}(X < b_l) \approx \mathrm{Prob}(X > b_r) \approx 0.05\,I_0\,\varepsilon_u$. Use recursion (10).
5: Compute $I \leftarrow \mathtt{AGL}(f, [b_l, b_r], tol = 0.05\,I_0\,\varepsilon_u)$.
[Store all calculated subintervals and their CDF values in a table.]

▷ Setup

6: Set $a_0 \leftarrow b_l$, $h \leftarrow (b_r - b_l)/128$, $F_0 \leftarrow 0$, and $k \leftarrow 0$.
7: **while** $a_k < b_r$ **do**
8: **loop** ▷ interpolating polynomial on $[a_k, a_k + h]$
9: Set $x_0 = 0, x_1, \ldots, x_n = h$ to rescaled Chebyshev points, see (4).
10: Set $u_0 \leftarrow 0$, compute $u_i \leftarrow u_{i-1} + \hat{I}_{(x_{i-1},x_i)}[f]$ for all $i = 1, \ldots n$.
[Reuse table from Step 5 together with simple Gauss-Lobatto.]
11: Compute coefficients $\{c_j\} \leftarrow \mathtt{NCoef}(\{u_j\}, \{x_j\})$.
12: Compute test points $\{t_j\} \leftarrow \mathtt{NTest}(\{u_j\})$.
13: Compute $\xi_i \leftarrow \mathtt{NEval}(\{c_j\}, \{u_j\}, t_i) \quad [= F_a^{-1}(t_i)]$ for all $i = 1, \ldots n$.
14: Compute $\varepsilon_i \leftarrow |\hat{I}_{(0,\xi_i)}[f] - t_i|$ for all $i = 1, \ldots, n$.
15: **if** $\max_{i=1,\ldots,n} \varepsilon_i \le \varepsilon_u$ and $x_{i-1} \le \xi_i \le x_i$ for $i = 1, \ldots, n$ **then**
16: Exit loop (goto line 20). ▷ u-error and monotonicity condition satisfied
17: **else**
18: Set $h \leftarrow 0.8\,h$ and try again (i.e. continue with line 9).
19: **end loop**
20: Set $h \leftarrow 1.3\,h$ if $\max \varepsilon_i \le \varepsilon_u/3$.
21: Store $\{c_j\}$, $\{u_j\}$, $\{x_n\}$, a_k, and F_k in table.
22: Set $h \leftarrow \min(h, b_r - (a_k - h))$ [take care of right boundary].
23: Set $k \leftarrow k + 1$, $a_k \leftarrow a_{k-1} + h$, and $F_k \leftarrow F_{k-1} + u_n$.
24: Create table for indexed search on $\{F_j\}$.

▷ Sampling

25: Generate $U \sim U(0, I)$.
26: Find interval J with $F_J \le U < F_{J+1}$ using indexed search.
27: Compute $X \leftarrow a_J + \mathtt{NEval}(\{c_J\}, \{u_J\}, U - F_J)$.
28: **return** X.

large) sample sizes the generation times of this R version is almost the same as for the C version. Our tests were performed on distributions of different shapes including Gaussian, Cauchy, beta, gamma, and t-distributions with various parameter settings. We also applied our algorithm successfully to several non-standard distributions including the generalized hyperbolic distribution [Barndorff-Nielsen and Blæsild 1983], the noncentral χ^2-distribution ([Fisher 1928], see [Johnson et al. 1995]) and the α-stable distribution ([Lévy 1925], see [Nolan 2010] for a recent

Table I. Required number of intervals for different u-resolutions ε_u using polynomials of order 1, 3 and 5, respectively.

ε_u	10^{-8}	10^{-10}	10^{-8}	10^{-10}	10^{-12}	10^{-8}	10^{-10}	10^{-12}
distribution	Order $n=1$		Order $n=3$			Order $n=5$		
Normal	12620	118294	173	517	1603	63	123	252
Cauchy	19512	193558	288	826	2504	112	203	393
Exponential	10914	108882	128	382	1192	44	87	176
Gamma(5)	11890	121602	177	526	1647	62	124	255
Beta(5,5)	11272	99702	155	477	1491	58	114	236
Beta(5,500)	11874	11130	178	527	1648	62	124	256

survey).

In extensive tests we observed that for all these distributions our algorithm results in approximations that have u-errors smaller than the one required by the user. u-resolutions of 10^{-12} were reached without problems for all the mentioned distributions. The set-up time is moderate and strongly depends on the time needed to evaluate the density f. The marginal execution times are very fast and practically the same for all tested distributions. The marginal execution times we observed were all faster than generating an exponential random variate by inversion.

The interested reader can find more information on the distributions we tested, on the results of our stability and accuracy tests and on the speed of the setup and of the sampling algorithm in the Online Supplement of this article.

5.1 The Required Number of Intervals

The required number of intervals is an important characteristic of the algorithm as it influences both the setup time and the size of the required table. Using the error-bound for interpolation which is $O(h^{n+1})$ for interval length h and order n it is obvious that the required number of intervals is $O(1/\varepsilon_u^{n+1})$. This implies that for linear interpolation an error-reduction by a factor of $1/100$ requires about ten times the number of intervals. Therefore, linear interpolation is not useful if small error values are required as the table sizes explode. For order $n = 3$ an error-reduction by a factor of $1/100$ requires $\sqrt{10} = 3.16$ times the number of intervals, for $n = 5$ this factor is reduced to $\sqrt[3]{10} = 2.16$. In Table I we report the required number of intervals for some standard distributions and practically important values of ε_u. These results clearly illustrate the asymptotic considerations for the required number of intervals. They also indicate that order $n = 5$ is enough to reach close to machine precision with a moderate number of intervals. Of course larger values of n are not desirable as they would lead to slower marginal execution times. Note that the speed differences between $n = 1$, $n = 3$ and $n = 5$ were close to negligible in our timing experiments.

The differences between distributions are not too large. The worst case of our examples is the Cauchy distribution whose heavy tails imply a large computationally relevant domain and thus many intervals. Otherwise the differences are small, monotone densities (like the exponential density) and densities without tail (like the Beta(5,5) density) require slightly less intervals than bell-shaped densities with two tails.

6. COMPARISON WITH UNIVERSAL ALGORITHMS OF THE LITERATURE

We have already pointed out that the automatic inversion algorithms described by Ahrens and Kohrt [1981] and Ulrich and Watson [1987] are based on similar ideas as our algorithm. They use a decomposition into many intervals and polynomial approximations of the inverse CDF within those intervals. The main difference to our algorithm is that all of their algorithms use a decomposition fixed at the beginning, whereas we select the decomposition such that the maximal error is below a user-selected threshold. Moreover, there is no further control of the resulting approximation error.

The algorithm by Ahrens and Kohrt [1981] is based on Lagrange interpolation on nine points in each of the intervals. The interpolating polynomials are then approximated by a truncated expansion into Chebyshev polynomials, the highest order polynomials are discarded as long as the sum of the absolute values of the neglected coefficients is below the required accuracy, and the remaining parts are reconverted into common polynomials. Thus the order of the approximating polynomials vary between different intervals to reach what they claim to be exact inversion for single precision. The cut-off points for the computational domain are found by a simple strategy. The algorithm requires the evaluation of the inverse CDF in the setup which is performed with a method that is related to Newton's root finding. The paper states that it "require[s] a double precision integration routine" for computing the CDF and that "iterated Simpson integration proved adequate".

Ulrich and Watson [1987] describe several numeric inversion methods. Besides using a packaged ODE solver they also developed fourth and fifth order Runge-Kutta type approximations to the inverse CDF based on the integration of the ODE $x'(u) = 1/f(x(u))$. All their algorithms are based on the same fixed subdivision of domain $(0, 1)$ as the method of Ahrens and Kohrt [1981]. A last approach presented there is the usage of B-splines which are available in many scientific software packages. Again the same fixed subdivision is used to compute a first approximation. This is then used to find near optimal knot placement for the final B-spline approximation. One of the appealing features of the B-spline approximation scheme is that the kth order B-spline will have a continuous $(k-2)$nd derivative and hence will provide a very smooth approximation to the inverse CDF. An obvious drawback is, that one cannot make local adaptive improvements of the approximation accuracy as in [Hörmann and Leydold 2003] or our new algorithm.

Ulrich and Watson [1987] report the x-errors of these algorithms. The maximal reported x-errors are all above 10^{-5} which cannot be called close to machine precision. They also report the maximal x-error of the algorithm of Ahrens and Kohrt [1981] which is always larger than 10^{-2} and mention that there is a problem with the precision in the tails. Both papers do not give a detailed description of the full algorithms and we were not able to get the original implementations. It is thus impossible for us to assess the speed and the exact size of the error of those algorithms.

Still we want to underline, that we consider the capability of the user to select the acceptable maximal u-error, which may be close to machine precision, as the main achievement of our new algorithm.

As linear interpolation, despite its unbeatable simplicity, is not capable to reach high precision with moderate tables (compare the required number of intervals in Table I), we compare our new algorithm only to our first numeric inversion algorithm HINV based on Hermite interpolation (see [Hörmann and Leydold 2003]; it is also implemented in our UNU.RAN library). A main difference to the new algorithm is that HINV requires the CDF and PDF for order $n = 3$ polynomials and also the derivative of the PDF for order $n = 5$; orders higher than 5 are not possible. A main reason for developing the new algorithm was that obtaining a precise implementation of the CDF is not easy for most important distributions. Combining it just with some numerical integrator when only the PDF is available was discouraging slow with, e.g., the generalized hyperbolic and the noncentral χ^2-distribution. Using the CDF allows a simpler cut-off procedure and avoids possible integration errors, but interestingly it does not improve the stability of the algorithm. Especially in the right tail the numerical instabilities of HINV are larger than those of our new algorithm. This is underlined by the fact that we observed several cases with u-errors larger than ε_u (about five percent of all cases we computed) when we tested the u-error of HINV in the tails of the distribution. For some parameter values of the t-distribution and $\varepsilon_u = 10^{-13}$ HINV is not able to reach the required accuracy and decomposes the domain into a huge number of intervals which never happened for our new algorithm. The numeric instabilities come from the fact that in the far right tail the CDF is only calculated with a precision of 10^{-16} and the probabilities of the intervals are small. Numeric integration in our new algorithm reduces that problem as we are calculating the CDF starting only with the left border of the current interval.

The marginal generation times of HINV and of the new algorithm are almost identical as the sampling algorithm is the same. The difference in the setup times is mainly caused by the relative speeds of the evaluations of the CDF and the PDF, respectively. In our experiments with the above distributions the setup of HINV was a bit faster than that of our new algorithm. For non-standard distributions with expensive PDFs the setup of the new algorithms is sometimes considerably faster than that of HINV. For the generalized hyperbolic distribution we observed that the setup of our new algorithm was about 100 times faster than that of HINV.

Another advantage of the new algorithm is that it requires only about half of the number of intervals to reach the same precision.

7. CONCLUSIONS

We have explained all principles and the most important details of a fast numeric inversion algorithm for which the user provides only a function that evaluates the density and a typical point in its domain. It is the first algorithm of this kind in the literature that is based on an error control, that works for all smooth bounded densities. Extensive numerical experiments showed that the new algorithm always reached the required precision for the Gamma, Beta and t-distribution and also for less well known distributions with computational difficult densities. For the fixed parameter situation our algorithm is by far the fastest inversion method known. Compared to the special inversion algorithms for the respective distributions we reached speed-up factors between 50 and 100 for the standard distributions and

above 1000 for important special distributions. This makes our algorithm in particular attractive for the simulation of marginal distributions, when using copula models, and for quasi-Monte Carlo applications.

ACKNOWLEDGMENTS

The authors gratefully acknowledge the useful suggestions of the area editor and two anonymous referees that helped to improve the presentation of the paper. The second author was supported by Boğaziçi-Research-Fund, Project 07HA301.

REFERENCES

Abramowitz, M. and Stegun, I. A., Eds. 1972. *Handbook of mathematical functions*, 9th ed. Dover, New York.

Ahrens, J. H. and Kohrt, K. D. 1981. Computer methods for efficient sampling from largely arbitrary statistical distributions. *Computing 26*, 19–31.

Barndorff-Nielsen, O. and Blæsild, P. 1983. Hyperbolic distributions. In *Encyclopedia of Statistical Sciences*, N. L. Johnson, S. Kotz, and C. B. Read, Eds. Vol. 3. Wiley, New York, 700–707.

Billingsley, P. 1986. *Probability and Measure*. Wiley & Sons, New York.

Bratley, P., Fox, B. L., and Schrage, E. L. 1983. *A Guide to Simulation*. Springer-Verlag, New York.

Chen, H. C. and Asau, Y. 1974. On generating random variates from an empirical distribution. *AIIE Trans. 6*, 163–166.

Cornish, E. A. and Fisher, R. A. 1937. Moments and cumulants in the specification of distributions. *Rev. de l'Inst. int. de stat. 5*, 307–322.

Costantini, P. 1986. On monotone and convex spline interpolation. *Mathematics of Computation 46,* 173, 203–214.

Dahlquist, G. and Björck, Å. 2008. *Numerical methods in scientific computing*. Vol. 1. SIAM, Philadelphia, PA.

Devroye, L. 1986. *Non-Uniform Random Variate Generation*. Springer-Verlag, New-York.

Fisher, R. A. 1928. The general sampling distribution of the multiple correlation coefficient. *Proceedings Royal Soc. London (A) 121*, 654–673.

Fisher, R. A. and Cornish, E. A. 1960. The percentile points of distributions having known cumulants. *Technometrics 2,* 2, 209–225.

Gander, W. and Gautschi, W. 2000. Adaptive quadrature–revisited. *BIT Numerical Mathematics 40,* 1, 84–101.

Hastings, C. 1955. *Approximations for Digital Computers*. Princeton University Press, Princeton, NJ, USA.

Hörmann, W. and Leydold, J. 2003. Continuous random variate generation by fast numerical inversion. *ACM Trans. Model. Comput. Simul. 13,* 4, 347–362.

Hörmann, W., Leydold, J., and Derflinger, G. 2004. *Automatic Nonuniform Random Variate Generation*. Springer-Verlag, Berlin Heidelberg.

Johnson, N. L., Kotz, S., and Balakrishnan, N. 1994. *Continuous Univariate Distributions*, 2nd ed. Vol. 1. Wiley, New York.

Johnson, N. L., Kotz, S., and Balakrishnan, N. 1995. *Continuous Univariate Distributions*, 2nd ed. Vol. 2. Wiley, New York.

Kennedy, W. J. and Gentle, J. E. 1980. *Statistical Computing*. STATISTICS: Textbooks and Monographs, vol. 33. Dekker, New York.

L'Ecuyer, P. 2008. *SSJ: Stochastic Simulation in Java*. Départment d'Informatique et de Recherche Opérationnelle (DIRO), Université de Montréal. Version 2.1.1 (Oct. 7, 2008).

Leobacher, G. and Pillichshammer, F. 2002. A method for approximate inversion of the hyperbolic cdf. *Computing 69,* 4, 291–303.

Lévy, P. 1925. *Calcul des Probabilités*. Gauthier-Villars, Paris.

Leydold, J. and Hörmann, W. 2009a. *Runuran – R interface to the UNU.RAN random variate generators, Version 0.10.1*. Department of Statistics and Mathematics, WU Wien, A-1090 Wien, Austria. `http://cran.r-project.org/`.

Leydold, J. and Hörmann, W. 2009b. *UNU.RAN – A Library for Non-Uniform Universal Random Variate Generation, Version 1.4.1*. Department of Statistics and Mathematics, WU Wien, A-1090 Wien, Austria. `http://statistik.wu.ac.at/unuran/`.

Monahan, J. F. 2001. *Numerical Methods of Statistics*. Cambridge University Press, Cambridge.

Nolan, J. P. 2010. *Stable Distributions – Models for Heavy Tailed Data*. Birkhäuser, Boston. In progress, Chapter 1 online at academic2.american.edu/∼jpnolan.

Overton, M. L. 2001. *Numerical Computing with IEEE Floating Point Arithmetic*. SIAM, Philadelphia.

R Development Core Team. 2008. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0.

Schwarz, H. R. and Klöckler, N. 2009. *Numerische Mathematik*, 7th ed. Vieweg+Teubner, Wiesbaden.

Thomas, D. B., Luk, W., Leong, P. H. W., and Villasenor, J. D. 2007. Gaussian random number generators. *ACM Computing Surveys 39,* 4.

Tuffin, B. 1997. Simulation accélérée par les méthodes de Monte Carlo et quasi-Monte Carlo: théorie et applications. Ph.D. thesis, Université de Rennes 1.

Ulrich, G. and Watson, L. 1987. A method for computer generation of variates from arbitrary continuous distributions. *SIAM J. Sci. Statist. Comput. 8*, 185–197.

Wichura, M. J. 1988. Algorithm AS 241: The percentage points of the normal distribution. *Appl. Statist. 37,* 3, 477–484.

Wilson, E. B. and Hilferty, M. M. 1931. The distribution of chi-square. *Proc. Nat. Acad. Sci. 17*, 684–688.